Sun Reference No.: P3689
Our Reference No.: 06502.0210

UNITED STATES PATENT APPLICATION

OF

Paul HINKER, Bradley LEWIS, and Michael BOUCHER

FOR

ADAPTIVE MEMORY ALLOCATION

FIELD OF THE INVENTION

This invention relates generally to data processing systems, and more particularly, to methods for optimizing the allocation and deallocation of shared memory to programs executing in a data processing system.

BACKGROUND OF THE INVENTION

During execution, programs typically make many dynamic memory access requests. Such requests involve requesting allocation of memory from a system call (e.g., malloc), utilizing the memory, and deallocating the memory using a system call (e.g., free). Dynamic memory allocation is a fundamental and very important part of many computer programs. It is therefore desirable to improve the performance of memory access functions.

SUMMARY OF THE INVENTION

In accordance with methods and systems consistent with the present invention, an improved memory access function (e.g., malloc) is provided that dynamically improves its performance by changing its operation at runtime. The memory access function adapts its operation based on previous memory access requests to be able to provide the fastest response to those kinds of memory access requests that it predicts will be the most common. For purposes of this description "access to system memory" includes both requests for allocation of system memory and release of system memory, or deallocation. Similarly, a "memory request" refers to either an allocation (a request for system memory), or a deallocation (a return of previously requested memory).

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

In accordance with methods and systems consistent with the present invention, as embodied and broadly described herein, a method is provided in a data processing system for allocating memory. The method receives a memory request for a reference to a block of memory. Then it returns the reference to the block of memory to satisfy the request. Next, it adjusts an operation of the memory access function based on the memory request.

Furthermore, in accordance with methods and system consistent with the present invention, as embodied and broadly described herein, a system for providing access to memory includes a memory which further includes a program including a memory access function that provides access to memory and adjusts its operation according to a memory request for a reference to a block of memory, and a processor for executing the program.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 depicts a block diagram of a data processing system suitable for practicing methods and systems consistent with the present invention;

Fig. 2 depicts in greater details the general access tree and the fast access tree depicted in Fig. 1;

Figs. 3A & 3B depict a flow diagram illustrating operations performed to allocate memory in accordance with the principles of the present invention; and

Fig. 4 depicts a flow diagram illustrating operations performed to deallocate memory in accordance with the principles of the present invention.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

DETAILED DESCRIPTION

Methods and systems consistent with the present invention provide a function called "smart-alloc" that adapts in real-time the memory allocation process to threads executing in a data processing system. More specifically, smart-alloc observes past memory allocation behavior and adjusts the process of providing memory access such that the most frequently requested memory blocks of a specific size are allocated faster than less frequently requested memory block sizes. Smart-alloc is a language-independent library function which may be linked to one or more programs operating in a multi-processing environment.

Overview

When a program begins execution, smart-alloc secures access to an amount of memory that may be accessed as needed by threads executing within the program. Upon receiving a pointer from a system memory call to an amount of memory requested, smart-alloc divides the memory into blocks of varying sizes, grouping blocks of like size together in a linked-list, referred to as a free memory linked-list. Smart-alloc creates two trees that point to the free memory linked-lists: a fast access tree and a general access tree. As a program executes, the fast access tree points to the most frequently accessed memory block sizes, and the general access tree points to block sizes not pointed to by the fast access tree. Each free memory linked-list will be pointed to by either the general access tree, or the fast access tree, but not both.

Upon receiving a request from an executing thread for access to memory, smart-alloc increments a counter associated with the block size that will be used to satisfy the memory request. To satisfy the memory request, smart-alloc first determines whether the fast access tree

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

points to memory blocks of the size needed to satisfy the request. If the fast access tree includes memory blocks of the size needed to satisfy the request, smart-alloc satisfies the request from the fast access tree. Otherwise, smart-alloc satisfies the request from the general access tree. If a request is satisfied from the general access tree, smart-alloc compares the value of the counter associated with the allocated block size to the counter values for block sizes included in the fast access tree to determine whether the general access tree refers to block sizes most frequently requested. If the counter value for the allocated block size (from the general access tree) is greater than any of the counter values of block sizes included in the fast access tree, smart-alloc replaces that general access tree pointer to the block size having the highest counter value with a pointer from the fast access tree, and replaces the pointer to the block size included in the fast access tree having the lowest counter value with a pointer from the general access tree. This process ensures that the fast access tree continues to point to the most frequently requested block sizes.

Smart-alloc also receives requests from executing threads desiring to release references to blocks of memory. After receiving a released reference, smart-alloc adds the reference to the free memory linked-list that includes memory blocks corresponding to the size of the returned reference.

Implementation Details

Fig. 1 depicts a block diagram of a data processing system 100 suitable for practicing methods and implementing systems consistent with the present invention. Data processing system 100 includes computer 102 connected to network 105. Computer 102 includes secondary

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

storage 110, processors 120a...n, output device 130, input device 140, and memory 150. Memory 150 includes programs 155, 160, and 165, operating system 170, and shared memory 180. Operating system 170 includes a system shared memory access function, malloc 182. Each of programs 155, 160, and 165 includes smart-alloc 185, a shared memory access function operating in accordance with the principles of the present invention. Each instance of smart-alloc 185 includes two data structures, a fast access tree 186 and a general access tree 187 that are used by smart-alloc 185 when allocating and deallocating memory.

Programs 155, 160, and 165 share access to shared memory 180. Programs 155, 160, and 165 may include a single thread or multiple threads. Although multiple processors are shown, one skilled in the art will appreciate that programs 155, 160, and 165 may execute on a single processor to implement methods and practice systems consistent with the present invention.

Operating system 170 represents the Solaris® operating system from Sun Microsystems, Inc., which specifically facilitates multi-threaded program execution, although any operating system may be used to implement methods and systems consistent with the present invention.

Although aspects of this implementation are depicted as being stored in memory 150, one skilled in the art will appreciate that all or part of systems and methods consistent with the present invention may be stored on or read from other computer readable media, such as secondary storage devices, like hard disks, floppy disks, and CD-ROM; a digital signal received from a network such as the Internet; or other forms of RAM or ROM, either currently known or later developed. Sun, Sun Microsystems, and the Sun logo are trademark or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

Fig. 2 depicts general access tree 202 and fast access tree 204 pointing to free memory linked-lists 262-272. In this example, general access tree 202 includes two main branches: branch 206 including pointers to free memory linked-lists containing references to memory blocks of a size less than or equal to 4K and branch 208 including pointers to free memory linked-lists containing references to memory blocks of a size greater than 4K. Branches 210, 215, and 225 serve as head pointers to free memory linked-lists 262, 266, and 272, which include memory blocks of size 1K, 4K, and 64K, respectively. Fast access tree 204 also includes two main branches, 230 and 240. Fast access tree 204 further includes head pointers 250, 255, and 260 to free memory linked-lists 264, 268, and 270, including memory blocks of size 2K, 8K, and 16K, respectively, indicating that these memory block sizes were most frequently accessed by the threads executing in the associated program. General access tree 202 may include additional branches that include head pointers to additional free memory linked-lists. Fast access tree 204, however, only includes head pointers to a specific number of free memory linked-lists, typically no more than eight. A "head pointer" is a pointer to the beginning of a free memory linked-list. Thus, because the fast access tree does not typically grow as large as the general access tree, accesses to the fast access tree are faster than accesses to the general access tree.

Figs. 3A & 3B depict a flow diagram of a smart-alloc function operating in accordance with the principles of the present invention. First, smart-alloc requests access to an amount of memory from a system memory call included as part of an operating system (step 305). If the amount of memory is available (step 310), the system memory call returns to smart-alloc a pointer to the amount of memory requested (step 315). Upon receiving a pointer to a requested amount of memory, smart-alloc divides the memory into blocks of varying sizes and builds a free

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

memory linked-list for each block size (step 318). The block sizes in the free memory linked-lists begin at 1K and increase in an amount equal to double the size of a previous block. For example, 1K, 2K, 4K, 8K, 16K, etc. Once the free memory linked-lists are created, smart-alloc creates the general access tree which includes head pointers to each of the free memory linked-lists (step 320). Initially, the fast access tree is empty. As a program executes and a series of memory requests are processed, the fast access tree is populated as discussed below.

Upon receiving a memory request from an executing thread (step 325), smart-alloc increments the value of a counter associated with the block size used to satisfy the request (step 330). Thus, if a thread requests 1026 bytes of memory and the free memory linked-lists include 1K and 2K blocks, a reference to the 2K block will be returned to the requesting thread, and the value of the counter associated with the 2K free memory linked-list will be incremented. If this memory request is satisfied from the fast access tree (step 335), processing ends.

Otherwise, if the memory request is satisfied from the general access tree (step 340), then the counter associated with the block size used to satisfy the request is compared with the counter for block sizes pointed to by the fast access tree (step 370). Smart-alloc then determines whether the block size should be added to the free memory linked-list pointed to by the fast access tree step 375 because it includes memory blocks of a size most often requested by the executing program. In this step, smart-alloc compares the value of the counter associated with a block size included in the general access tree with the value of the counter of a block size included in the fast access tree. If the counter value for the block in the general access tree is greater than the counter value for the block in the fast access tree, the pointer from the general access will be replaced with a pointer from the fast access tree. Additionally, the fast access tree pointer that

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

points to the block size having the lowest counter value is replaced with the pointer from the general access tree (step 380). Changing the pointers in this manner ensures that the fast access tree continually points to the most frequently allocated block sizes, thereby increasing the efficiency of memory allocation. Otherwise, if the counter value associated with the block in the general access tree is not greater than the counter value of any of the free memory linked-lists pointed to by the fast access tree, processing ends.

If a memory request cannot be satisfied from either the general or fast access trees, smart-alloc requests additional memory from a system memory call (step 345). If the additional memory is available (step 350), a pointer to the amount of memory is returned to smart-alloc (step 355), the memory is divided into blocks that are added to the free memory linked-lists pointed to by a pointer from the general access tree (step 360), and processing continues to step 335. Otherwise, if the requested amount of memory is not available, the system memory call returns a null pointer to smart-alloc (step 365), and processing ends.

Fig. 4 depicts a flow diagram of operations performed by smart-alloc when a thread releases a reference to a block of memory. Upon receiving a reference to a block of memory from an executing thread (step 410), smart-alloc adds the reference to the appropriate free memory linked-list (step 430) and processing ends.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

Conclusion

Methods and systems operating in accordance with the principles of the present invention optimize the process of allocating memory to threads executing in programs operating in a data processing system by adapting the allocation process according to the behavior of an executing program. This smart-alloc function is a library function that may be linked to any program executing in a data processing system. As discussed above, smart-alloc operates in the user space and does not require accessing the operating system each time memory is allocated or deallocated. These distributive and adaptive features of smart-alloc allow it to minimize the number of accesses to the operating system, a costly and time consuming event, and ultimately yields increased system performance.

Methods and systems consistent with the present invention are applicable to all single and multi-threaded programs written in all computer programming languages, including Fortran 77, Fortran 90, Java, C, and C++. Although maximum increases in efficiency will be realized in a multi-processor computing environment, methods and systems consistent with the present invention may also provide operational benefits in a single or multi-threaded, single processor environment.

Although the foregoing description has been described with reference to a specific implementation, those skilled in the art will know of various changes in form and detail which may be made without departing from the spirit and scope of the present invention as defined in the appended claims and the full scope of their equivalents.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000